Agenda

Simulating gates, circuits, and storage

- HDL to C/C++
- C/C++ review

Lab – Working with an integrated circuits (IC)

- Getting started with breadboards

Simulating gates and other hardware

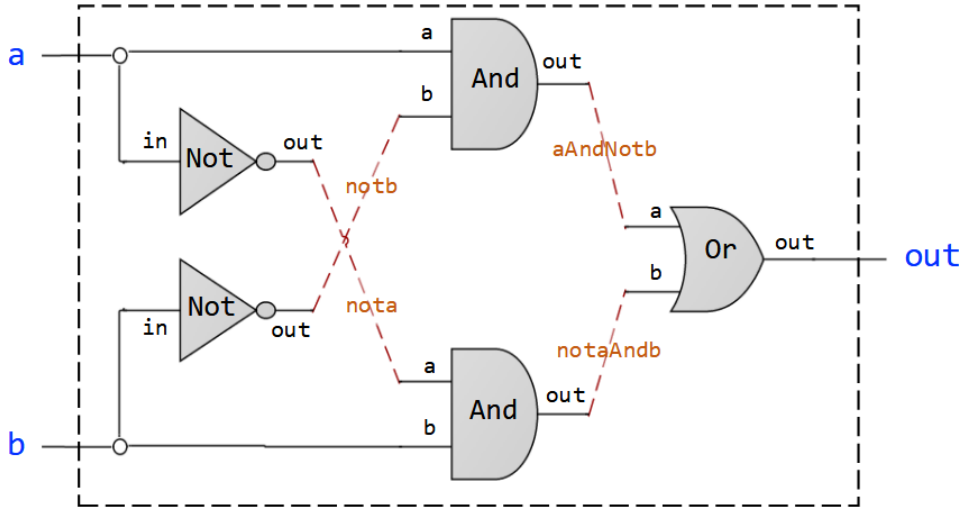
Idea: Simulate hardware designs before building them

- Design the chip architecture
- Encode the design with HDL
- Simulate the chip and optimize the design
- Manufacture the physical chip

HDL – Hardware Description Language

VHDL – Virtual Hardware Description Language

HDL Demo: XOR Gate




```
/** out = (a And Not(b)) Or (Not(a) And b)  */
CHIP Xor {
    IN a, b;
    OUT out;

    PARTS:
    Not (in=a, out=nota);
    Not (in=b, out=notb);
    And (a=a, b=notb, out=aAndNotb);
    And (a=nota, b=b, out=notaAndb);
    Or  (a=aAndNotb, b=notaAndb, out=out);
}
```

```
/** Chips set (APIs): */
...
Not (in=, out= );
And (a=, b=, out= );
Or  (a=, b=, out= );
Xor (a=, b=, out= );
...
```

From nand2tetris.org

C Demo: HDL to C

```
CHIP Xor {
   IN a, b;
   OUT out;

   PARTS:
   Not(in=a, out=nota);
   Not(in=b, out=notb);
   And(a=a, b=notb, out=w1);
   And(a=nota, b=b, out=w2);
   Or(a=w1, b=w2, out=out);
}
```

```
void Xor(uchar a, uchar b, uchar* out)
{
  uchar nota, notb, w1, w2;
  Not(a, &nota);
  Not(b, &notb);
  And(a, notb, &w1);
  And(nota, b, &w2);
  Or(w1, w2, out);
}
```

Goal: Build all gates from Nand

Write code to test inputs/outputs

Review: Pass by value vs Pass by Pointer

```
typedef unsigned char uchar;

void Xor(uchar a, uchar b, uchar* out)
{
  uchar nota, notb, w1, w2;
  Not(a, &nota);
  Not(b, &notb);
  And(a, notb, &w1);
  And(nota, b, &w2);
  Or(w1, w2, out);
}

int main()
{
  uchar x, y, result;
  Xor(x, y, &result);
  return 0;
}
```

Hardware simulator design (part 1)

Use functions to simulate chips with no persistent state

Elementary logic gates

- Not
- And
- Or
- Xor
- Mux
- DMux

16-bit variants

- Not16
- And16
- Or16
- Mux16

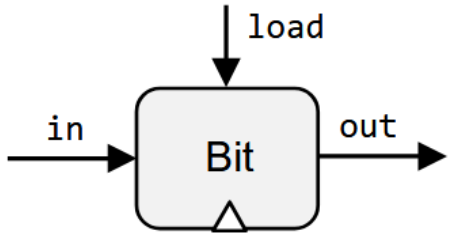
Multi-way variants

- Or8Way
- Mux4Way16
- Mux8Way16
- DMux4Way
- DMux8Way

Operations

- HalfAdder
- FullAdder
- Add16
- Inc16
- ALU

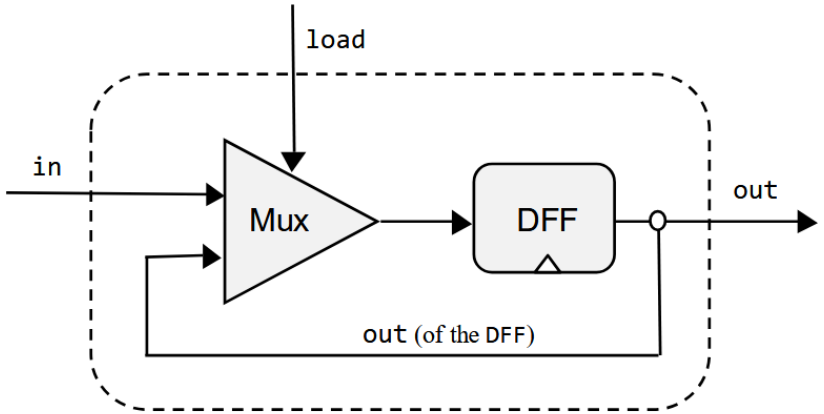
Simulating memory



1-bit register

<u>1-bit register</u>

Stores one bit over time



if load(t):
  out($t+1$) = in(t)
else
  out($t+1$) = out(t)

Simulating a 1-bit DFF in C++

```
// register.h
namespace hack
{
  class Bit // one-bit data flip-flop
  {
    public:
      Bit();
      void tick();
      uchar in;
      uchar load;
      uchar out;
  };
}
```

```
// register .c
namespace hack
{
 Bit::Bit() : load(0), out(0), in(0) {}
  void Bit::tick()
  {
    Mux(out, in, load, &out);
    load = 0;
  }
}
```

```
int main()
{
  hack::Bit b;
  b.in = 1;
  b.load = 1;
  b.tick();
  return 0;
}
```

NOTE: namespaces help to avoid name conflicts

Defining classes in C++

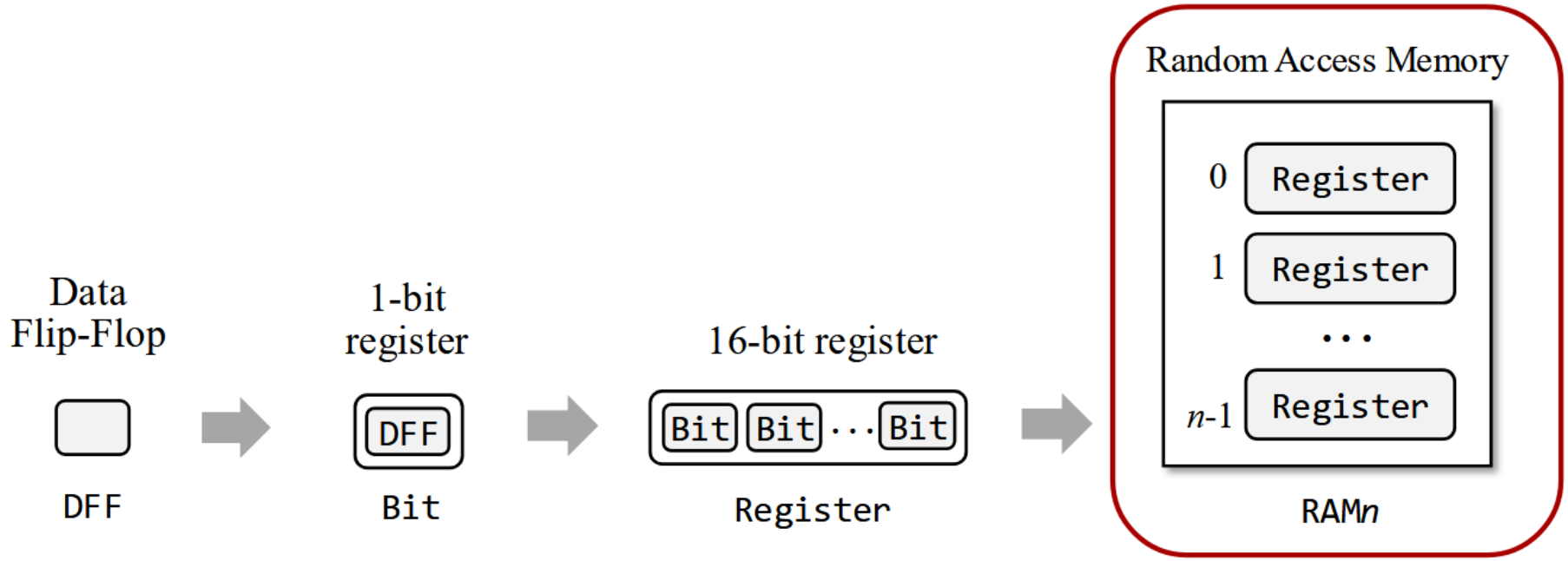
```
class Box {
  protected float mySize = 1.0f;
  public Box(float s) {
     mySize = s;
  }

  public float getSize() {
     return mySize;
  }
}
```

```
class Box {
 public:
  Box(float s) {
     mySize = s;
  }
 float getSize() {
     return mySize;
 }
 protected:
  float mySize = 1.0f;
};
```

Simulating RAM

Build up memory recursively from smaller components



Simulating RAM of different sizes

```
template <class T, int k>
class RAM
{
  public:
    RAM() : load(0) { Zero16(in); Zero16(out); }
    void tick()
    {
    }

    uchar in[16];
    uchar out[16];
    T M[8];
    uchar address[16];
    uchar load;
};

typedef RAM<Register,3> RAM8;
typedef RAM<RAM8,6> RAM64;
```

```
int main()
{
  hack::RAM8 ram8;

  dec2bin(9, ram8.in);

  // Set address to 2
  ram8.address[0] = 0;
  ram8.address[1] = 1;
  ram8.address[2] = 0;
  ram8.load = 1;
  print_memory(&ram8); ram8.tick();
  print_memory(&ram8);
  return 0;
}
```

Simulating RAM of different sizes

```
template <class T, int k>
class RAM
{
  public:
    RAM() : load(0) { Zero16(in); Zero16(out); }
    void tick()
    {
    }

    uchar in[16];
    uchar out[16];
    T M[8];
    uchar address[16];
    uchar load;
};

typedef RAM<Register,3> RAM8;
typedef RAM<RAM8,6> RAM64;
```

Testing RAM8

```
int main()
{
  hack::RAM8 ram8;

  dec2bin(9, ram8.in);

  // Set address to 2
  ram8.address[0] = 0;
  ram8.address[1] = 1;
  ram8.address[2] = 0;
  ram8.load = 1;
  print_memory(&ram8); ram8.tick();
  print_memory(&ram8);
  return 0;
}
```

namespaces

Avoids name conflicts
In .c files, the using keyword can be used to simplify names from a namespace

```
// register.h
namespace hack
{
  class Bit // one-bit data flip-flop
  {
    public:
      Bit();
      void tick();
      uchar in;
      uchar load;
      uchar out;
  };
}
```

```
#include “register.h”

int main()
{
  hack::Bit bit;
  return 0;
}
```

```
#include “register.h”
using namespace hack;

int main()
{
  Bit bit;
  return 0;
}
```

Hardware simulator design (part 2)

Use classes for components with persistent state

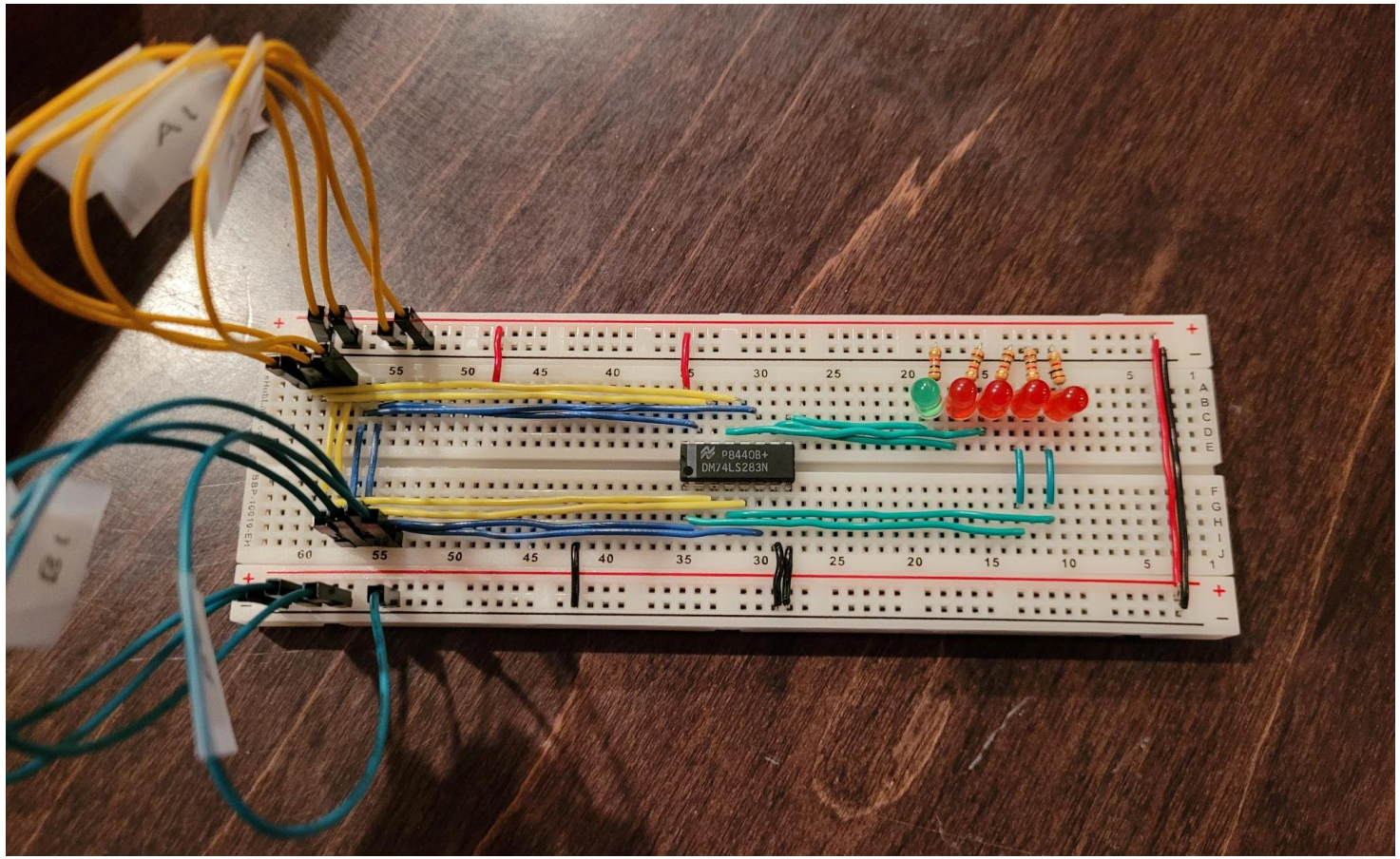
Components:

- Bit
- Register
- Counter
- RAM

Lab – Working with an integrated circuit (IC)

An integrated circuit, or chip, combines electronic components.

Goal: Using a breadboard, we will visualize the output of either a 4-bit adder or xor using LEDs.



Background: circuits

A circuit is a circular flow of electricity.

Electricity wants to flow from high voltage (+) to low voltage (-)

- Power sources have two sides
  - Positive (+): denoted with red color
  - Negative (-): called ground and denoted with black or blue color

Watch out: never plug the positive voltage source directly into the ground voltage source (short circuit)

Watch out: if there is no connected loop between the voltage source and ground, nothing will happen (open circuit)

https://learn.sparkfun.com/tutorials/what-is-a-circuit

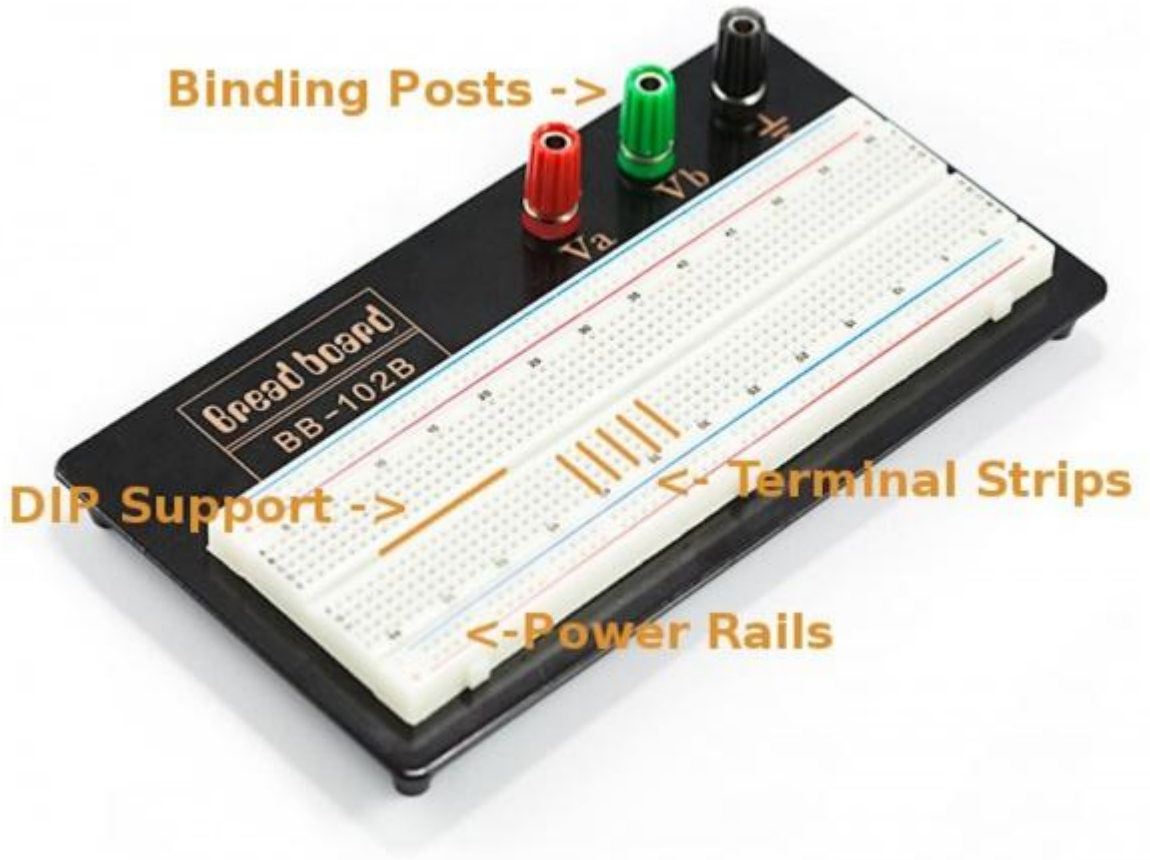
Getting started with breadboards

Breadboards allow you to prototype circuits without soldering

We plug in the board using the binding posts

Chips are placed in the middle (DIP support)

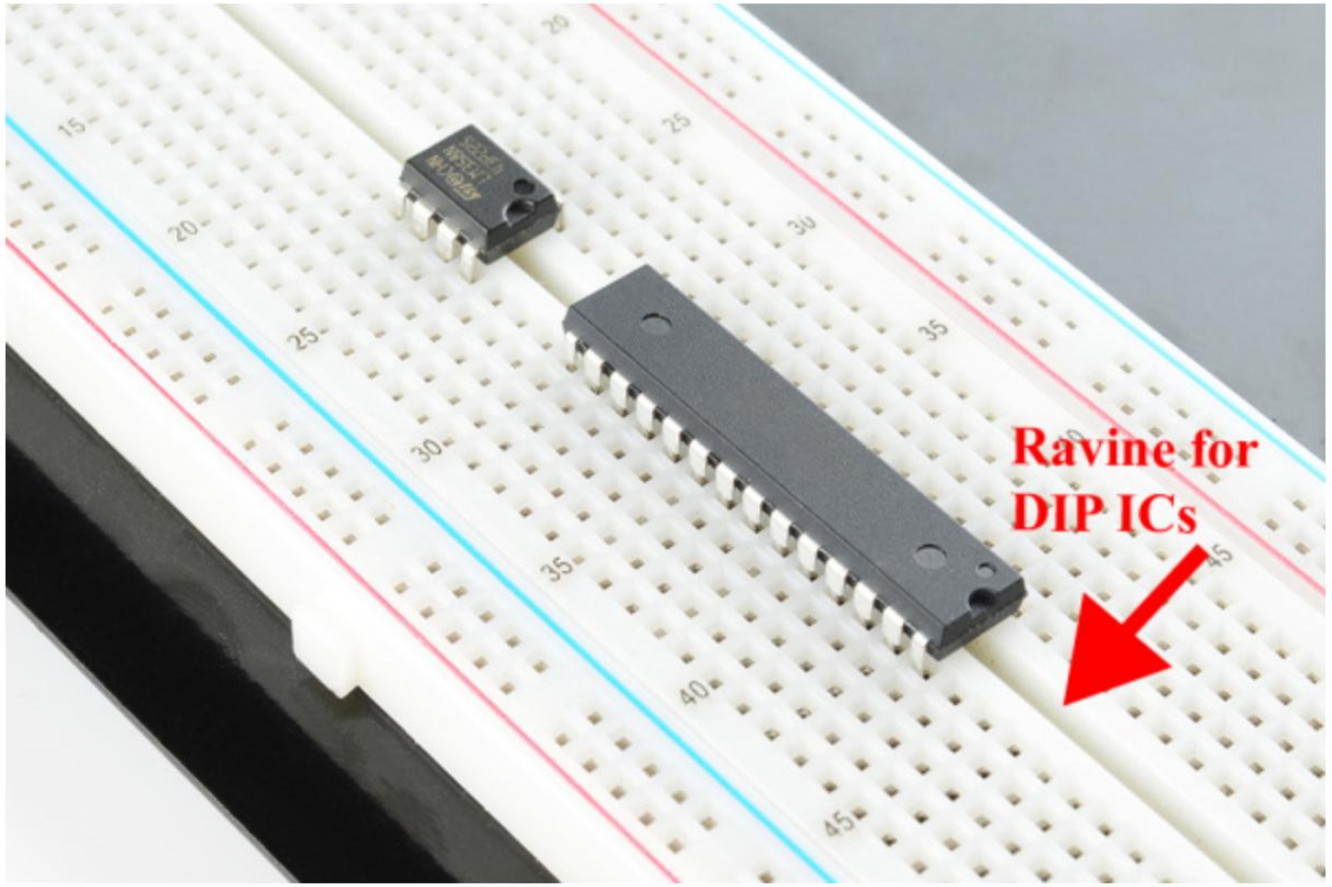
Power is siphoned from the power rails to components/pins located on different terminal strips



https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard

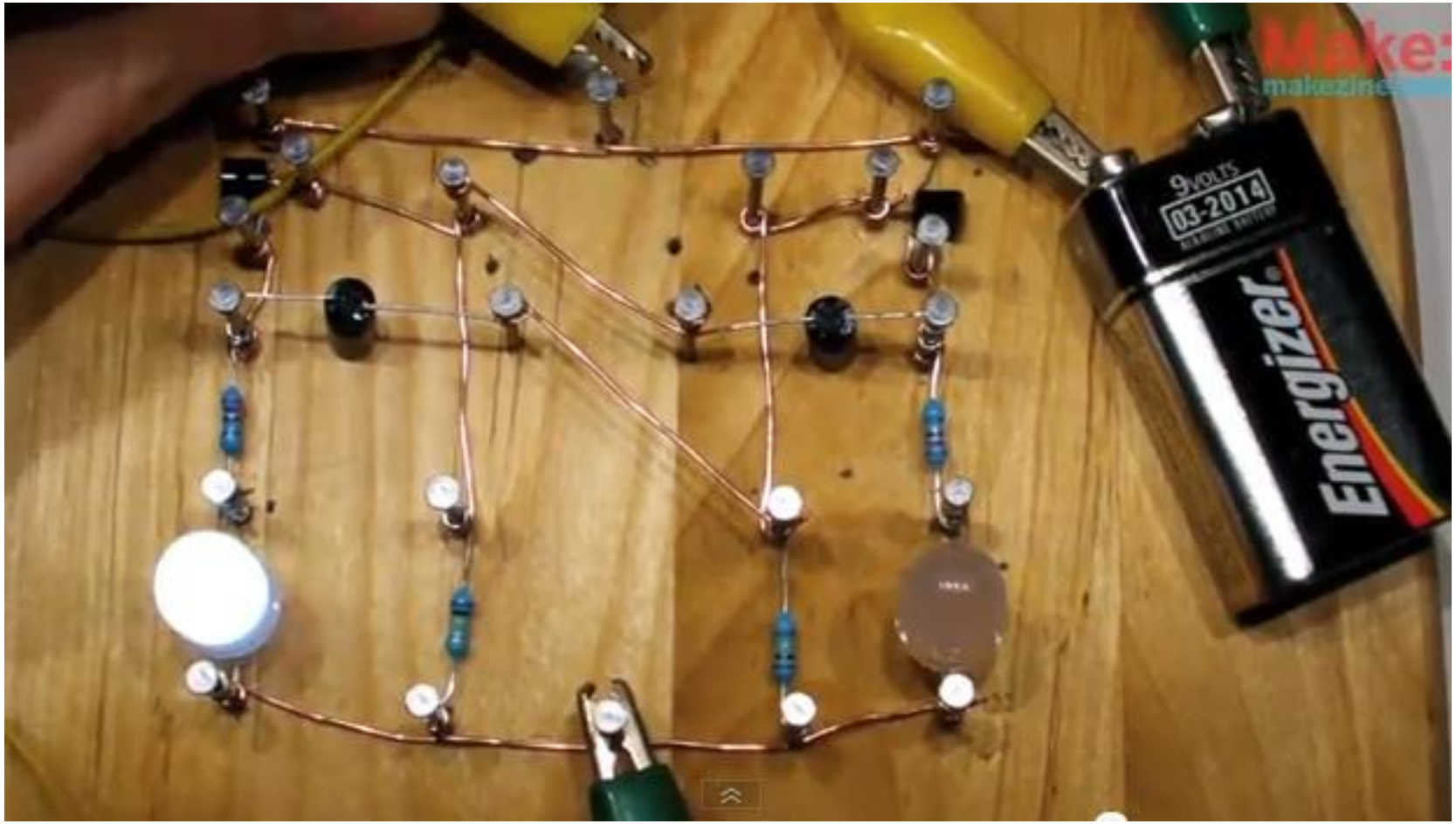
Placing ICs on a breadboard

Dual In-line Package (DIP) refers to ICs that sit over the middle of the breadboard.

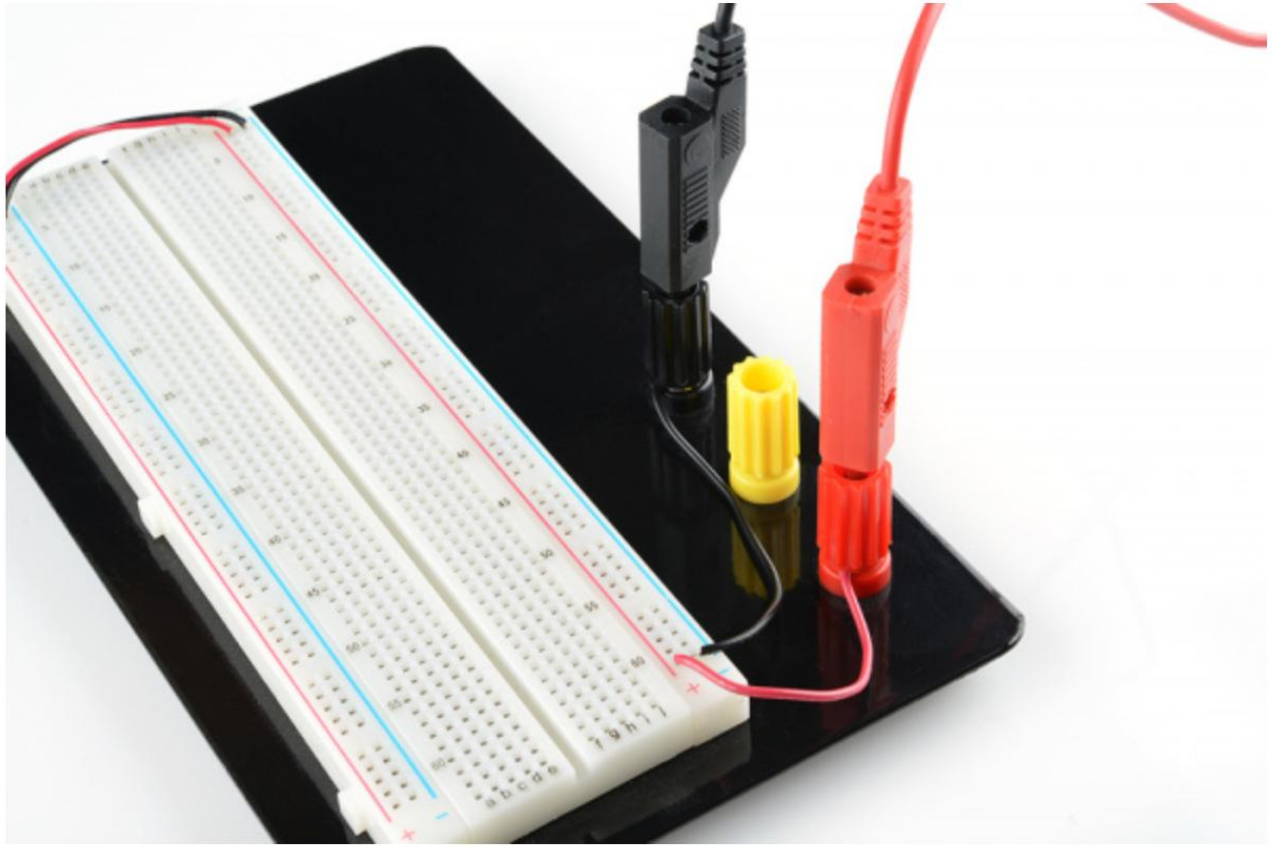


Aside: Why are these called breadboards?

Early engineers would prototype circuits by wrapping wires around nails in a wooden board, like those used to cut bread

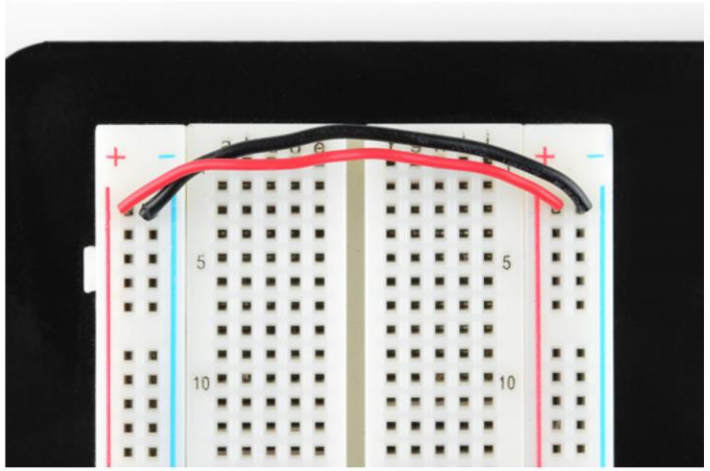


Powering the breadboard



A breadboard being powered through the binding posts from banana cables.

Binding Post for Banana Cables and Wires



Two jumper wires used to connect the power rails on both sides. Always attach the '+' to '+' and the '-' to '-'.

Breadboard safety

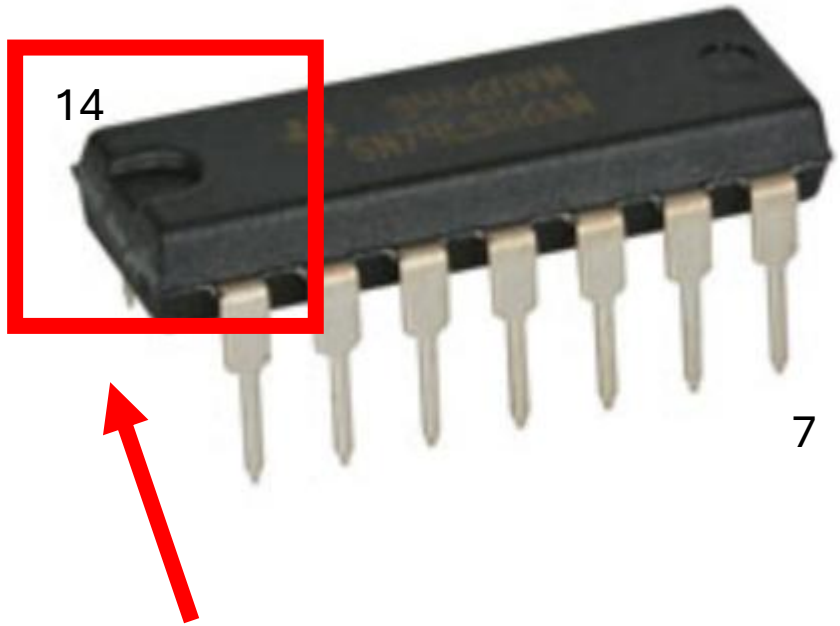
We will be working with 5V current.

1. Always work on your circuit with the power disconnected, and check your circuit carefully before plugging it in.
2. If you see smoke coming out of your circuit, or one of the circuit components becomes very hot, unplug the microcontroller immediately.
3. Never connect power pins (5 V, 3.3V, Vin) to ground (GND).
4. Always use color conventions for power wires (red = power, black = ground).
5. Keep wires as short as possible and avoid crossing wires or components over each other.
6. Keep water away from your circuits as possible.

Breadboard safety

5V is a low current and the risk is low. However, be particularly careful of the following

- MAKE SURE that the power (VCC) and ground pins of your circuit are connected to the correct power rails
  - A mistake can both burn out the chip and your fingers
  - ALWAYS use red for power and black for ground
- ALWAYS place a resistor with your LEDs
  - A mistake can burn our the LED



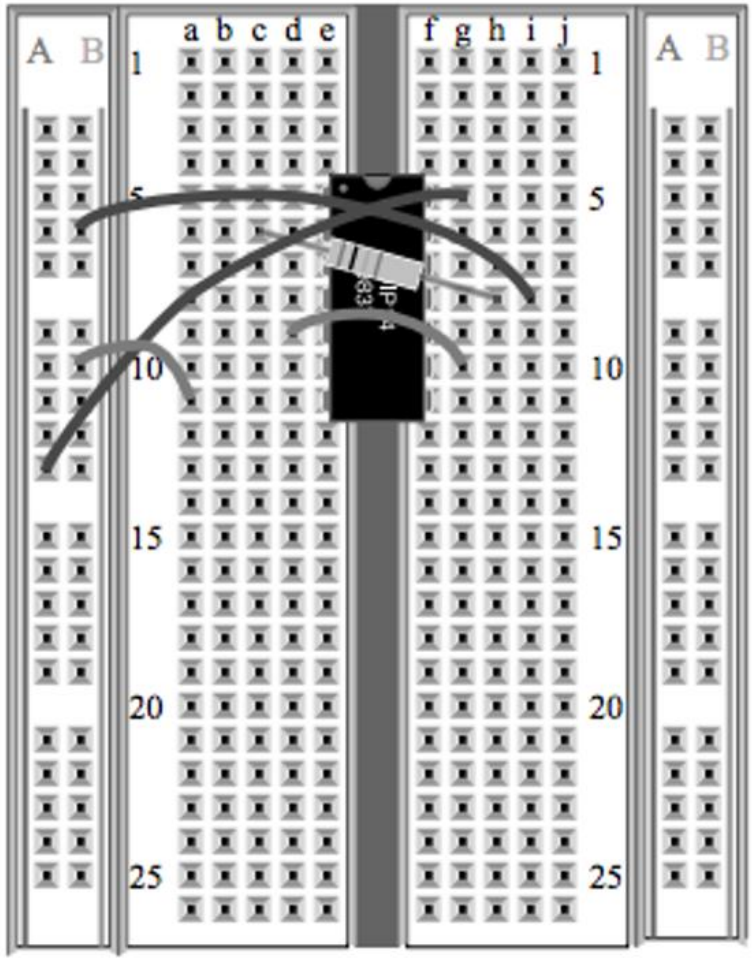
The half-moon notch can be used to orient the chip.

Breadboard best practices

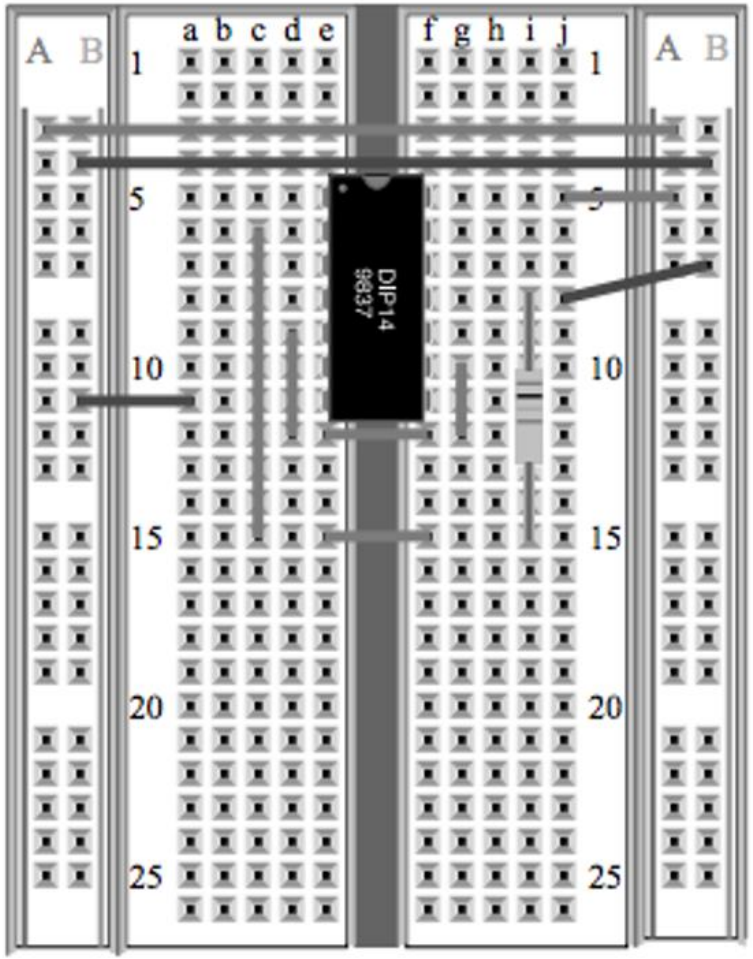
Be Tidy

Keep the design simple so you can follow the connections of your circuits

Don’t criss-cross wires and components



(a)



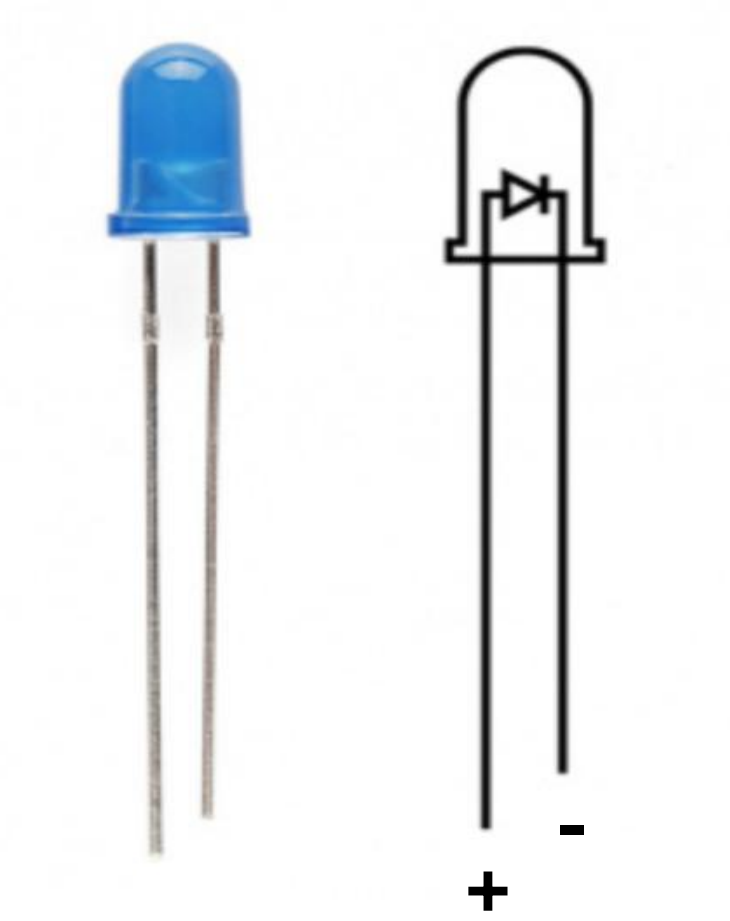
(b)

LEDs (Light-Emitting Diodes)

LEDS have a positive leg and a negative leg

Electricity will only flow through the LED in the right direction

LEDs will burn out if the current is too high. Always place a resistor next to the LED.



Resistors

Resistors resist the flow of electricity.

Use resistors to protect sensitive components like LEDS

The strength of the resistor is measured in ohms and indicated by the colored stripes on the resistor.

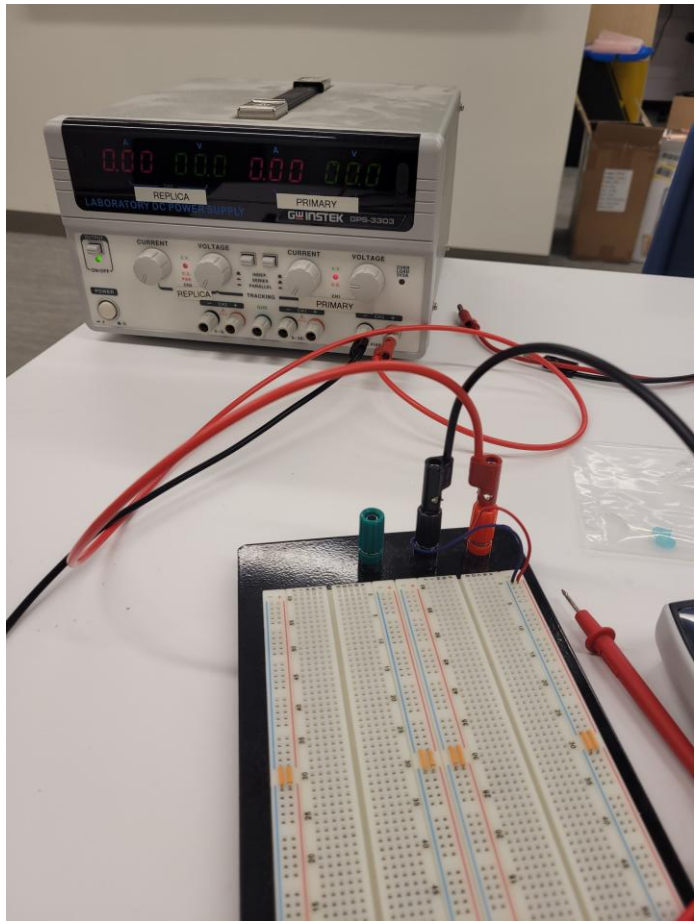
Step 1: Hook up power

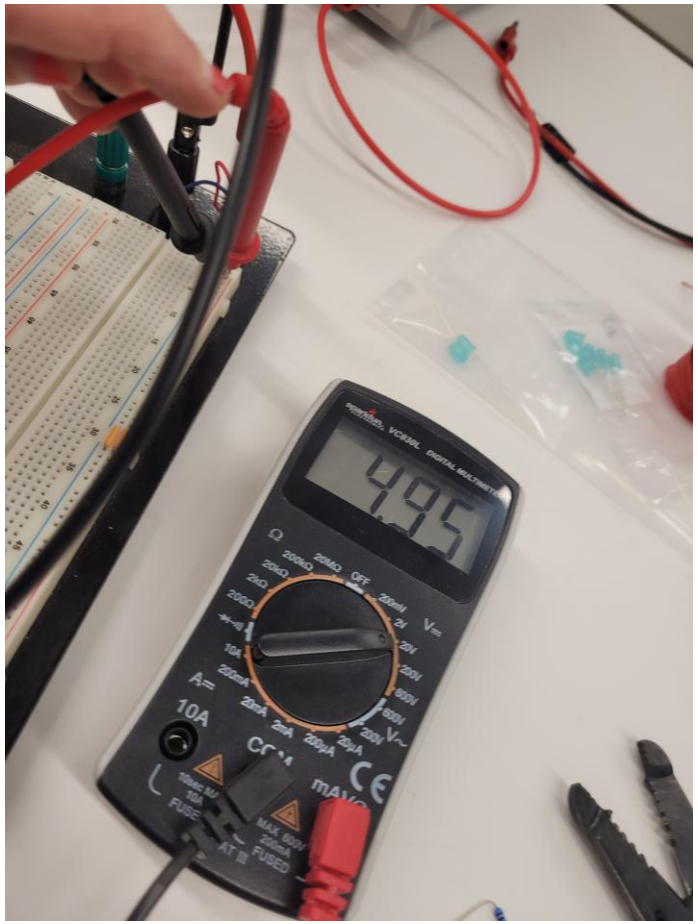
Plug-in your breadboard with the 5V power source.

Connect the binding posts to the power rail

WATCH OUT to always connect positive to positive and negative to negative.

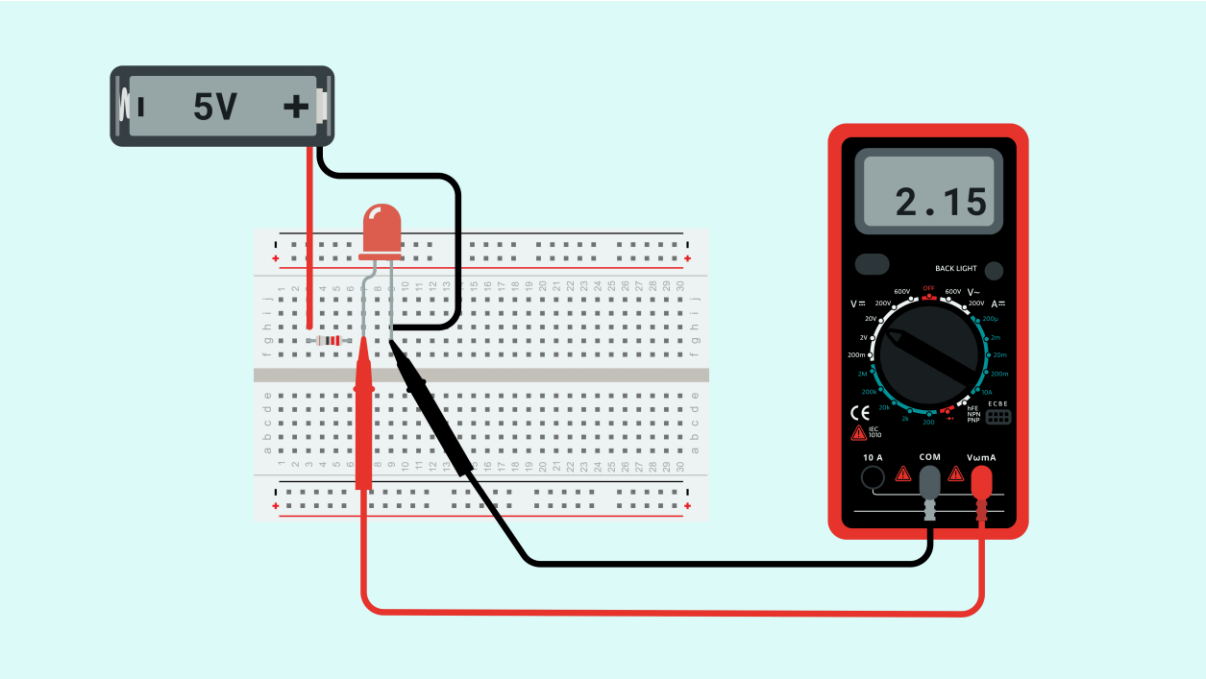
Use a multimeter that the power is setup correctly.

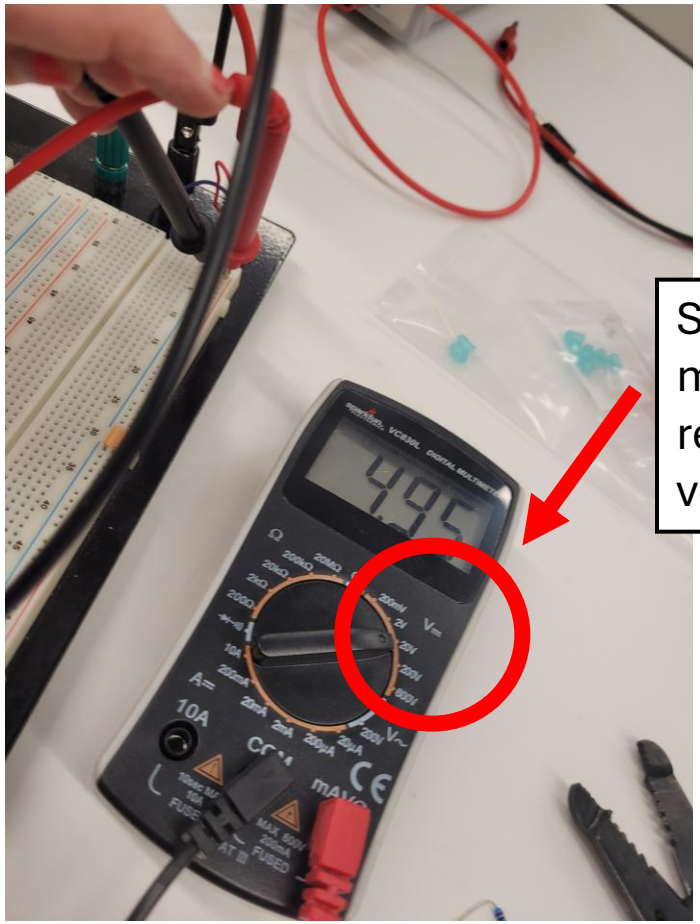




Aside: Reading voltage with a multimeter

A **multimeter** is tool for measuring voltage and resistance and is useful for debugging circuits.



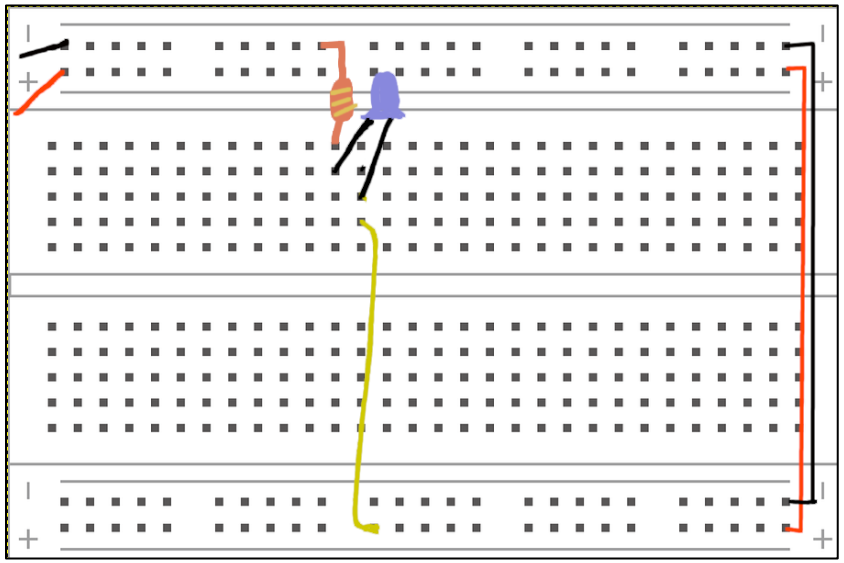


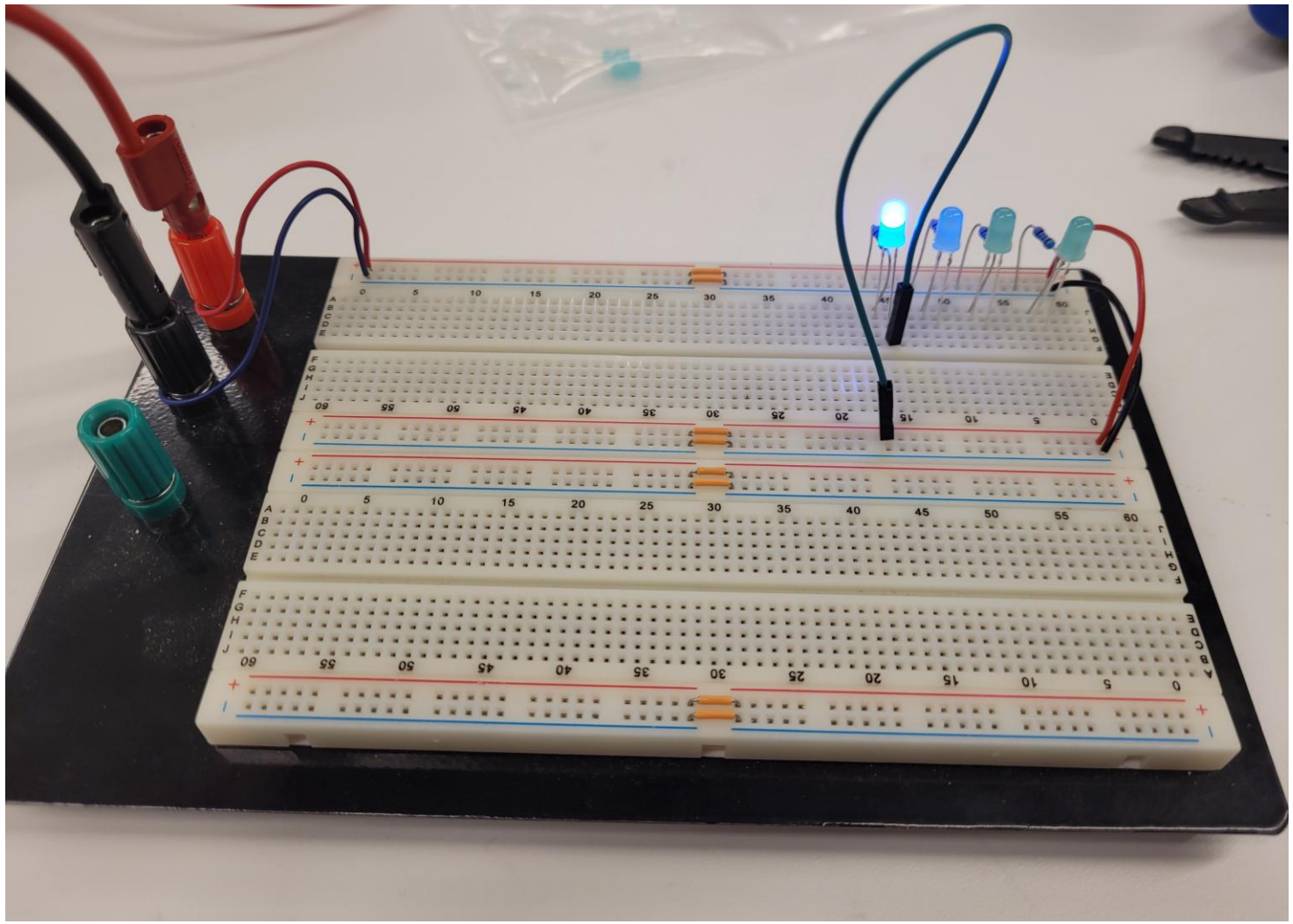
https://docs.arduino.cc/learn/electronics/multimeter-basics/

Step 2: Add LEDs and connect both power rails

2a. Connect the other power rail

2b. Add your LEDs and resistors.





Step 3: Setup and test your IC

3a. Use the data sheet for your chip to get the pin layout. See next slides.

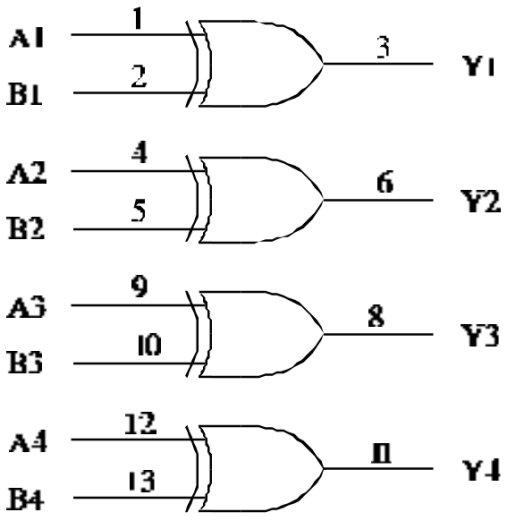
3b. Connect the chip to power (be careful here!)

3c. The input pins are not in order. Wire the inputs so that the A value is on one side of the board and the B values on the other.

3d. Connect the outputs to the LEDS. Again, watch out for the pin order!

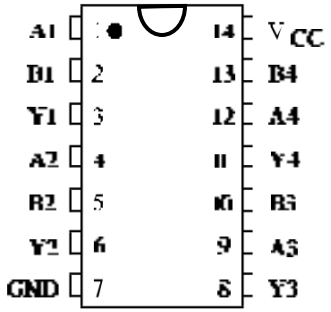
XOR: SL74LS86

LOGIC DIAGRAM



PIN 14 = V_{CC}
PIN 7 = GND

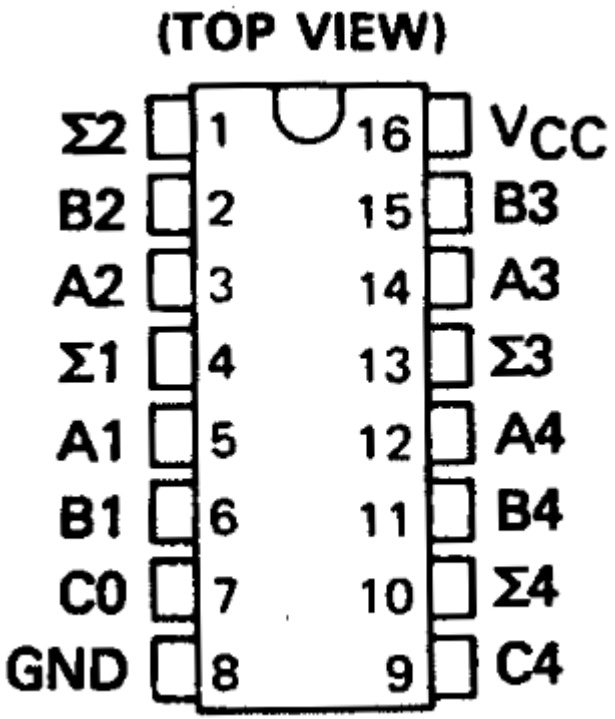
PIN ASSIGNMENT



FUNCTION TABLE

Inputs		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

Adder: 74LS283



FUNCTION TABLE

INPUT				OUTPUT					
				WHEN C0 = L			WHEN C0 = H		
				WHEN C2 = L			WHEN C2 = H		
A1 / A3	B1 / B3	A2 / A4	B2 / B4	Σ1 / Σ3	Σ2 / Σ4	C2 / C4	Σ1 / Σ3	Σ2 / Σ4	C2 / C4
L	L	L	L	L	L	L	H	L	L
H	L	L	L	H	L	L	L	H	L
L	H	L	L	H	L	L	L	H	L
H	H	L	L	L	H	L	H	H	L
L	L	H	L	L	H	L	H	H	L
H	L	H	L	H	H	L	L	L	H
L	H	H	L	H	H	L	L	L	H
H	H	H	L	L	L	H	H	L	H
L	L	L	H	L	H	L	H	H	L
H	L	L	H	H	H	L	L	L	H
L	H	L	H	H	H	L	L	L	H
H	H	L	H	L	L	H	H	L	H
L	L	H	H	L	L	H	H	L	H
H	L	H	H	H	L	H	L	H	H
L	H	H	H	H	L	H	L	H	H
H	H	H	H	L	H	H	H	H	H